MODELE

DJANGO 02

CZYM W OGÓLE TAK NAPRAWDĘ SĄ MODELE

- Wspomniałem na poprzednim wykładzie, że w modelach dodajemy do naszej strony poszczególne elementy. To pewne uproszenie
- DEFINICJA - model to typ obiektu, który reprezentuje dane i relacje w bazie danych
- Tak naprawdę głównym zadaniem modeli jest dodawanie, modyfikowanie oraz usuwanie danych bez konieczności pisania kodu SQL!

KROK 1

- Utworzenie klasy dla modelu
- W odróżnieniu od czystego pythona przy definicji klasy nie używamy self - nie odnosi się do siebie, lecz do zewnętrznej biblioteki - models.
- Wybór elementów, które będziemy używać - w pierwszej kolejności konstruowanie prostych modeli kojarzy się z tworzeniem formularza. Czy elementów prostej www

KROK 1 - PRZYKŁAD

```python
# models.py
from django.db import models

class Post(models.Model):
    title = models.CharField(max_length=100)
    text = models.TextField()
    author = models.ForeignKey(User, on_delete=models.CASCADE)
```

- Klasyczny import
- W klasie znajdują się elementy. Można zakończyć zwykłym () ale nie ustalimy wówczas żadnych ograniczeń
- Coś takiego będzie trzeba zmigrować do bazy danych.
- Ale wpierw jeszcze zarejestrujmy nasz model w Django

KROK 2 – DODANIE MODELU DO APLIKACJI, W ZASADZIE APLIKACJI DO DJANGO

- Trzeba dodać każdą aplikację do listy zainstalowanych - INSTALLED_APPS w pliku settings.py
- Może to być zwykły wpis jako nazwa - ’nazwa_aplikacji’ - django samo tworzy pliki konfiguracyjne
- A można od razu pełną konfigurację - ‘nazwa.apps.NazwaConfig’

SKĄD WZIĄĆ MODELE – ELEMENTY

- Większość jest dość intuicyjna (przynajmniej po 15 latach z php/html itp)
- Oczywiście najwłaściwszym byłoby przeczytać, mieć pod ręką dokumentację biblioteki. Pomagają różne czaty.
- Część rzeczy naprawdę się zapamiętuje, ale trochę przykładów podam

PRZYKŁADY PÓL W MODELU

- CharField - pole tekstowe - jak zaczniemy wpisywać parametr np max_lenght - to sam podpowie
- IntegerField - pole na liczby
- DateTimeField - data i czas - można precyzyjnie format ustalać
- I wiele wiele innych. Zasada użycia - nazwa_pola = models.TypPola(parametry)

PRZYKŁADY PÓL W MODELU

- **ForeignKey** służy do tworzenia relacji „wiele do jednego” między modelami w Django. Aby użyć pola ForeignKey, musisz podać dwa argumenty pozycyjne: model, do którego odwołuje się pole, i on_delete, który określa, co zrobić z obiektami powiązanymi, gdy obiekt nadrzędny zostanie usunięty

```
from django.db import models

class Author(models.Model):
    name = models.CharField(max_length=50)

class Book(models.Model):
    title = models.CharField(max_length=100)
    author = models.ForeignKey(Author, on_delete=models.CASCADE) # usuwanie obiektu
książki po usunięciu obiektu autora
```

CZYM SĄ WIDOKI I JAK JE POWIĄZAĆ Z MODELAMI

- Views w Django to funkcje lub klasy, które odpowiadają za obsługę żądań HTTP i wyświetlanie odpowiedzi HTTP. Views są związane z modelami przez to, że mogą odbierać dane z modelu jako parametry lub jako atrybuty obiektu widoku

```
from django.db import models

class Author(models.Model):
    name = models.CharField(max_length=50)

class Book(models.Model):
    title = models.CharField(max_length=100)
    author = models.ForeignKey(Author, on_delete=models.CASCADE) # usuwanie obiektu
książki po usunięciu obiektu autora
```

CZYM SĄ WIDOKI I JAK JE POWIĄZAĆ Z MODELAMI

- Można powiedzieć, że views określa zachowania i interakcje aplikacji webowej

```
from django.http import HttpResponse
from .models import Book

def book_list(request):
    # pobieramy wszystkie książki z bazy danych
    books = Book.objects.all()
    # tworzymy listę książek do przekazania do szablonu
    book_list = books.values_list('title', 'author', 'price')
    # wracamy do szablonu book_list.html
    return render(request, 'book_list.html', {'book_list': book_list})
```

```
def post_list(request):
    return render(request, 'blog/post_list.html', {})
```

PRZYPOMINAM O URLS

- W URLS dodajemy dowiązania do naszych aplikacji. W dużym skrócie. Jest to ściśle powiązane z Views. URLS jest miejscem, gdzie możemy ustawić mapowania url do widoku. A w views ustawiamy co ma się uruchomić po skorzystaniu z url.

projekt

```
1
2  from django.contrib import admin
3  from django.urls import path, include
4  #from pacjenci.views import test_response
5  urlpatterns = [
6      path('admin/', admin.site.urls),
7      path('pacjenci/', include('pacjenci.urls')) #to prowadzi do urls w aplikacji
8  ]
9
```

```
1  from django.urls import path
2  from pacjenci.views import spis_pacjentow
3  urlpatterns = [
4
5      path('pacjenci/', spis_pacjentow)
6  ]
```

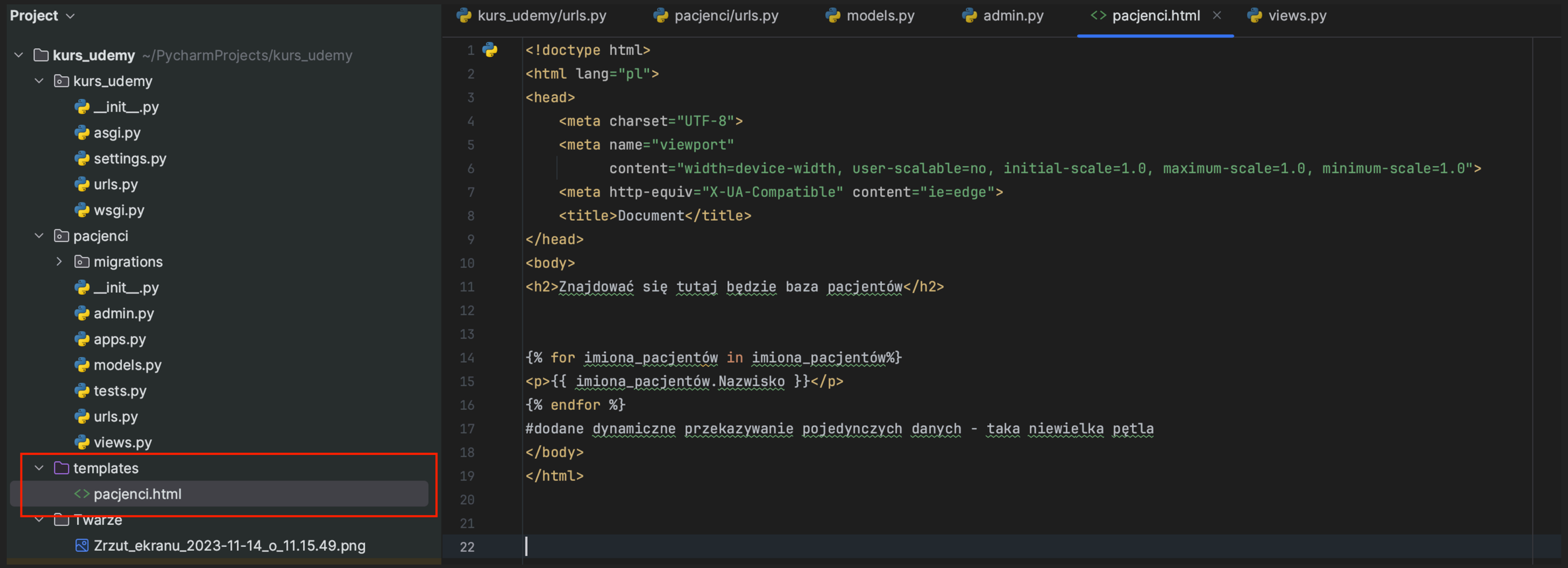
- W dużym uproszczeniu - odwiedzenie ścieżki pacjenci wywołuje funkcję spis_pacjentów

SKORO WIDOK ZWRACA, CO MA SIĘ POJAWIĆ PO SKORZYSTANIU Z URL

```python
from django.shortcuts import render
from django.http import HttpResponse

def test_response(request):
    return HttpResponse("To jest nasz pierwszy test")

```

```python
from django.contrib import admin
from django.urls import path
from filmyweb.views import test_response

urlpatterns = [
    path('admin/', admin.site.urls),
    path('test/', test_response)
]

```

CZYLI WŁĄCZAMY DO TEGO WSZYSTKIEGO TEMPLATES



CZYLI WŁĄCZAMY DO TEGO WSZYSTKIEGO TEMPLATES

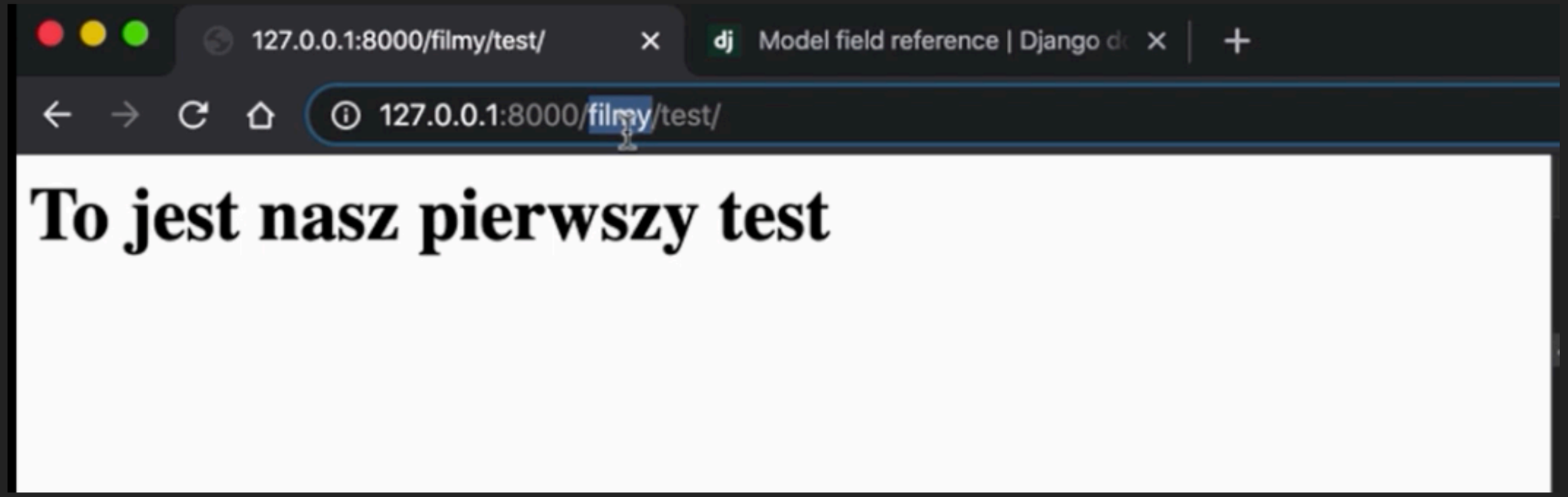
Zasadnicze pytanie, czy projekt w ogóle widzi nasze szablony. Jak korzystamy ze wspomagania w Pycharm Pro, to z automatu jest to w ustawieniach. W razie W trzeba znaleźć w settings odpowiednią linię i dodać wpis

```
54
55  TEMPLATES = [
56      {
57          'BACKEND': 'django.template.backends.django.DjangoTemplates',
58          'DIRS': [BASE_DIR / 'templates']
59          ,
60          'APP_DIRS': True,
61          'OPTIONS': {
62              'context_processors': [
63                  'django.template.context_processors.debug',
64                  'django.template.context_processors.request',
65                  'django.contrib.auth.context_processors.auth',
66                  'django.contrib.messages.context_processors.messages',
67              ],
```

CZYLI WŁĄCZAMY DO TEGO WSZYSTKIEGO TEMPLATES

Dobrze zawczasu stworzyć urls dla aplikacji osobno. Przykład już był.

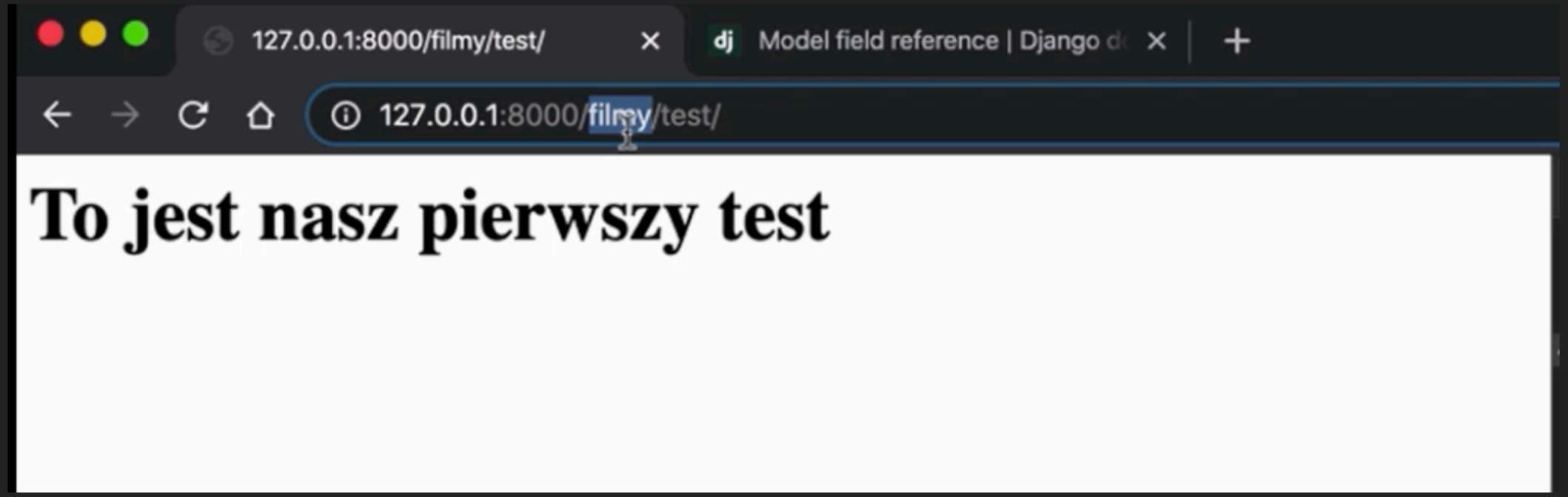
Później łatwiej zarządzać aplikacjami / widokami



CZYLI WŁĄCZAMY DO TEGO WSZYSTKIEGO TEMPLATES

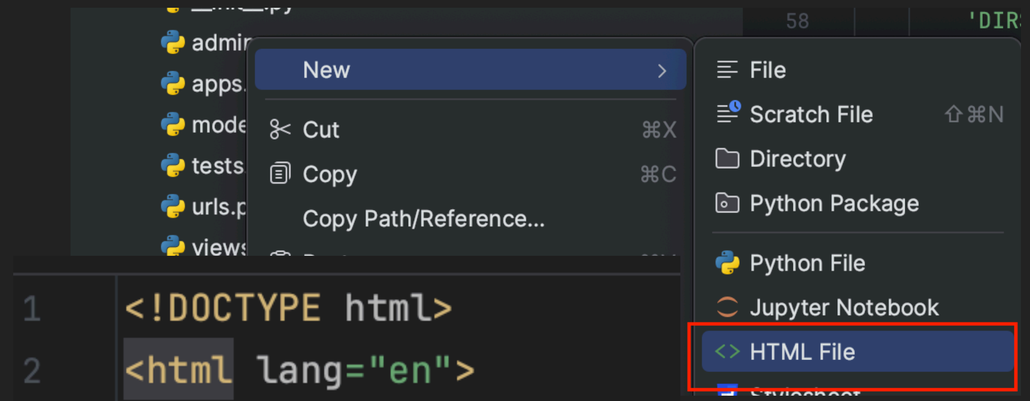
Dobrze zawczasu stworzyć urls dla aplikacji osobno. Przykład już był.

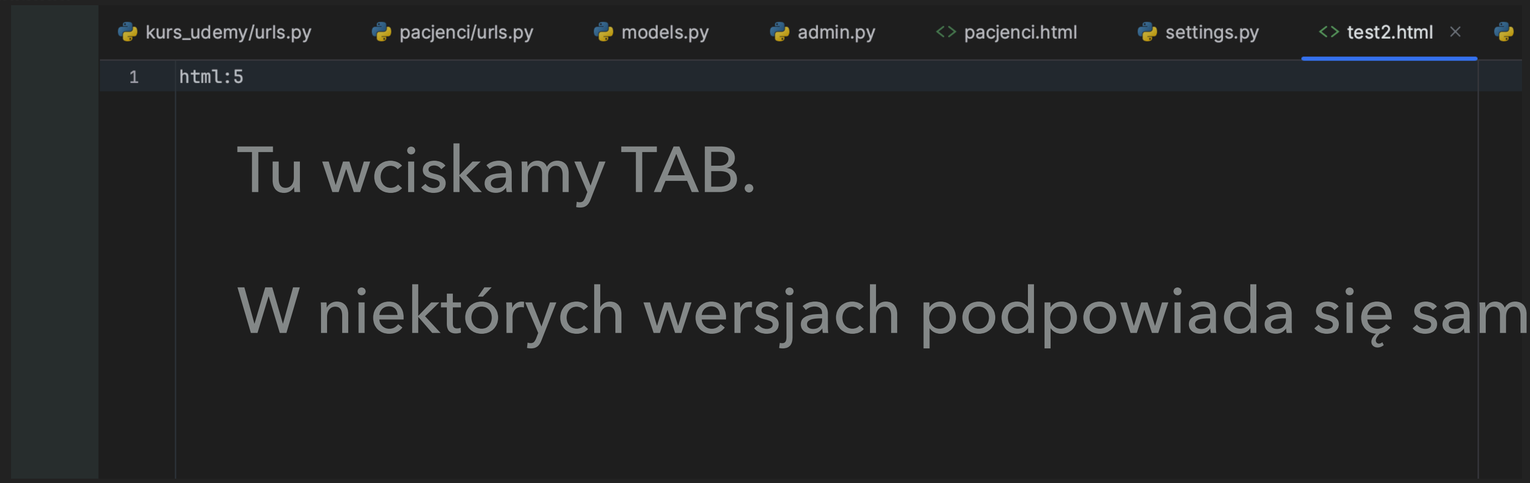
Później łatwiej zarządzać aplikacjami / widokami



CZYLI WŁĄCZAMY DO TEGO WSZYSTKIEGO TEMPLATES

W folderze Templates tworzymy sobie nasze szalony do widoków, w formacie html:





CZYLI WŁĄCZAMY DO TEGO WSZYSTKIEGO TEMPLATES

Trochę na poważniej zabieramy się za views - oczywiście w aplikacji - podłączamy szablon

```python
from django.shortcuts import render
from django.http import HttpResponse


def test_response(request):
    # return HttpResponse("<h1>To jest nasz pierwszy test</h1>")
    return render(request, 'filmy.html')
```

MOŻE ZAWARTOŚĆ ZROBIĆ NIECO BARDZIEJ DYNAMICZNIE

Chodzi o przekazywanie danych z aplikacji do html. Najpierw widok - na razie w jednym pliku

```
from django.shortcuts import render
from django.http import HttpResponse

def wszystkie_filmy(request):
    test = "To jest cos we views"
    return render(request, 'filmy.html', {'text': test})
```

MOŻE ZAWARTOŚĆ ZROBIĆ NIECO BARDZIEJ DYNAMICZNIE

Później podłączamy to do html. Do tego aby podłączyć taką właśnie dynamiczną tre
używa się czegoś, co się nazywa syntaksami - {{ text }} - to co się znajduje w klamrac
to jest dynamiczny kontent. To zostało dynamicznie z views przekazane

```python
from django.shortcuts import render
from django.http import HttpResponse

def wszystkie_filmy(request):
    test = "To jest cos we views"
    return render(request, 'filmy.html', {'text': test})
```

```
<p>{{ film }}</p>
```

MOŻE ZAWARTOŚĆ ZROBIĆ NIECO BARDZIEJ DYNAMICZNIE

W sumie aby wyświetlić nieco więcej warto zrobić coś na kształt pętli.

Pętla jakby sprawdzała czy coś się nie zmienia.

```
{% for imiona_pacjentów in imiona_pacjentów%}
<p>{{ imiona_pacjentów.Nazwisko }}</p>
{% endfor %}
#dodane dynamiczne przekazywanie pojedynczych danych - taka niewielka pętla
```

Pętla dla każdej zmiennej tworzy też nowy paragraf - fajnie to ustawia

MOŻE ZAWARTOŚĆ ZROBIĆ NIECO BARDZIEJ DYNAMICZNIE – JESZCZE BARDZIEJ – ORM

Możemy sobie zaimportować obiekty z naszej bazy danych

```
def spis_pacjentow(request):
    wszyscy = Pacjent.objects.all()
    return render(request, template_name: 'pacjenci.html', context: {'imiona_pacjentów': wszyscy}) #wyścietla obiekty z bazy danych
```

Tym razem mamy naprawdę dynamiczną zawartość